

Taller: Desarrollo de MODs para Minecraft

Microsoft Developer eXperience

Contenido

Introducción.....	2
Creación del proyecto.....	2
Construcción del bloque personalizado.....	3
Personalización del bloque	10
Renombrado del bloque	12
Recetas.....	14
Aún hay más.....	15

Introducción

En este taller vamos a ver las bases para crear Mods para Minecraft desde Visual Studio utilizando la API de Minecraft Forge.

Requisitos previos:

Es necesario tener instalado lo siguiente para seguir el tutorial correctamente:

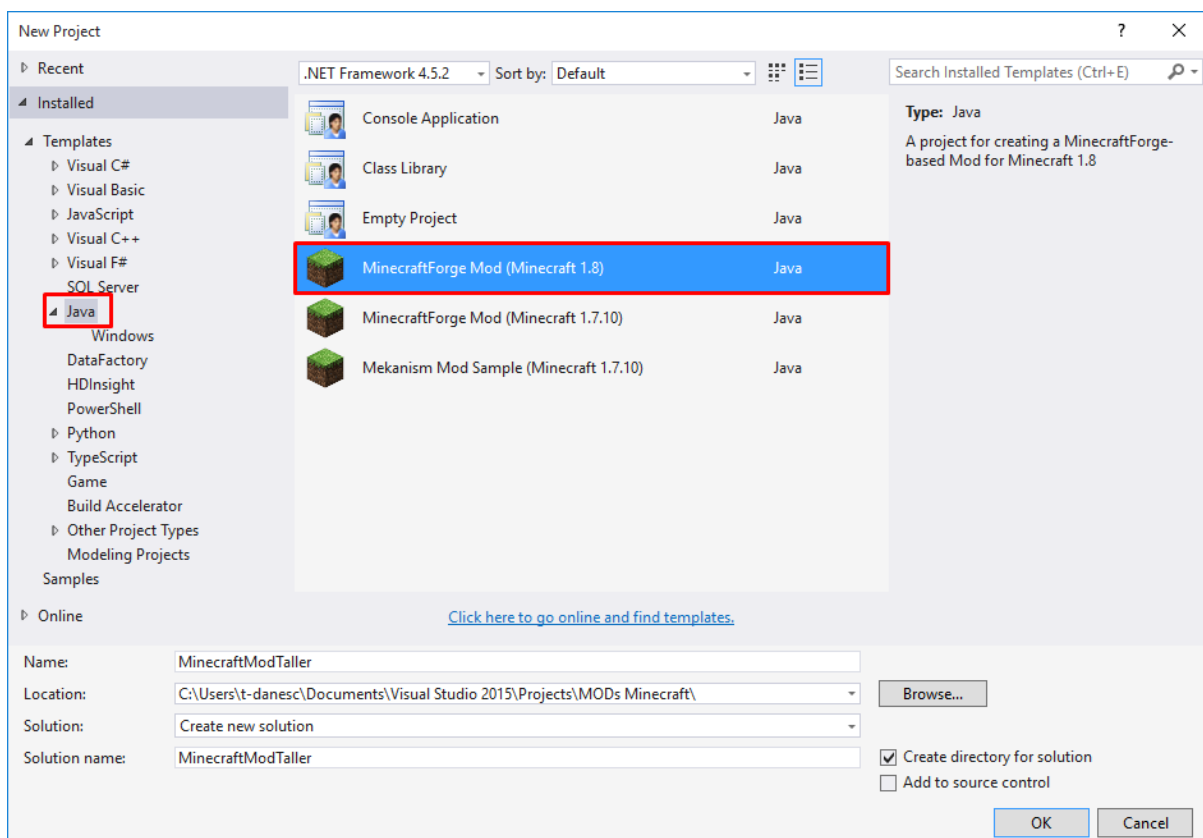
- Visual Studio 2013 (con la versión Community vale).
- [Java SE Development Kit \(JDK\)](#) para Windows x64 (**debe ser la versión de 64 bits**)
- Minecraft Mod Developer Pack
- Minecraft

Creación del proyecto

Tras la instalación del software necesario, empezaremos creando un proyecto. El ejemplo de este taller consistirá en la creación de un nuevo bloque que esté disponible para utilizar dentro del juego.

Para empezar, crearemos un nuevo proyecto. Al haber instalado el Minecraft Mod Developer Pack, nos aparecerán nuevas plantillas.

Abrimos Visual Studio y vamos a *File* → *New* → *Project...*, y buscamos de las plantillas *Java* la denominada *MinecraftForge Mod (Minecraft 1.8)*.



Tras pulsar en *Ok*, se nos creará el proyecto y se nos abrirá un fichero advirtiéndonos que Visual Studio empezará a descargar librerías necesarias para hacerlo funcionar correctamente. De hecho, si observamos la ventana *Output*, veremos que se ha iniciado una Build automáticamente.

```

Output
Show output from: Build
----- Build started: Project: MinecraftModTaller, Configuration: DebugClient Any CPU -----
Setting up MinecraftForge workspace... (this may take a while but it's a one-time build)
Note: This extension will download and build third party packages. Each third party package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, this
gradlew setupDecompWorkspace --refresh-dependencies

```

Esperamos a que finalice (puede llevar varios minutos). Estará finalizado cuando aparezca lo siguiente:

```

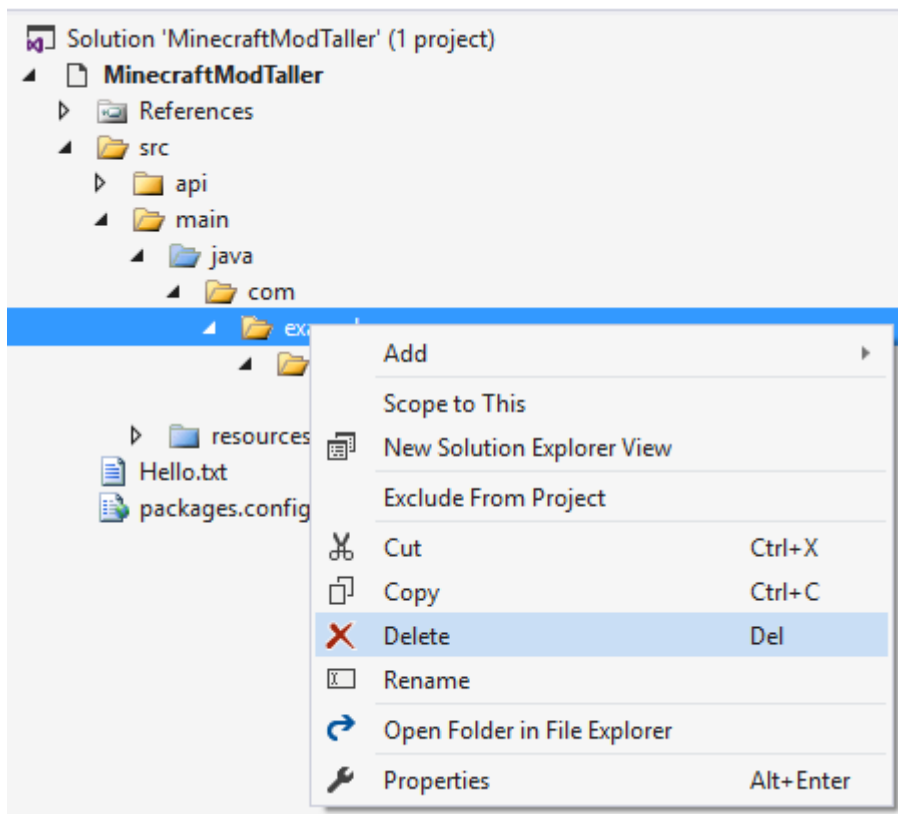
Output
Show output from: Build
:extractMinecraftSrc SKIPPED
:recompMinecraft SKIPPED
:repackMinecraft SKIPPED
:setupDecompWorkspace
BUILD SUCCESSFUL
Total time: 5 mins 53.241 secs
Setting up Eclipse workspace...
gradlew eclipse
Powered By MCP:
http://modcoderepack.com/
Searge, ProfMobius, Fesh0r,
R4wk, ZeuK, Ingiskahn, bspkrs
MCP Data version : snapshot_20141130
: eclipseClasspath
: eclipseJdt
: eclipseProject
: eclipse
BUILD SUCCESSFUL
Total time: 16.479 secs
Done!
Importing references:
C:/Users/t-danesc/.gradle/caches/minecraft/net/minecraftforge/forge/1.8-11.14.1.1357/snapshot/20141130/forgeSrc-1.8-11.14.1.1357.jar;C:/Users/t-danesc/.gradle/caches/modules-2/files-2.1/net.minecraf
C:/Program Files/Java/jdk1.8.0_73/bin/javac.exe -verbose -g -d obj/DebugClient/ -Xlint -deprecation -cp C:/Users/t-danesc/.gradle/caches/minecraft/net/minecraftforge/forge/1.8-11.14.1.1357/snapshot/
MinecraftModTaller -> C:/Users/t-danesc/Documents/Visual Studio 2015/Projects/MODs/Minecraft/MinecraftModTaller/MinecraftModTaller/bin/mods/MinecraftModTaller.jar
***** Build: 1 succeeded or up-to-date, 0 failed, 0 skipped *****
C:/Users/t-danesc/.gradle/caches/minecraft/net/minecraftforge/forge/1.8-11.14.1.1357/snapshot/20141130/forgeSrc-1.8-11.14.1.1357.jar
C:/Users/t-danesc/.gradle/caches/modules-2/files-2.1/net.minecraft/launcher/1.11/9c0592c6e1e9ea296a70948081bd4cc84dda1289/launcher-1.11.jar
C:/Users/t-danesc/.gradle/caches/modules-2/files-2.1/org.ow2.asm/asm-debug-all/5.0.3/f9e364ae2a66c2a543012a4668856e84e5da074/asm-debug-all-5.0.3.jar
C:/Users/t-danesc/.gradle/caches/modules-2/files-2.1/com.typesafe.akka/akka-actor_2.11/2.3.3/ed65e9fc709ca0f2f1a3210daa0b70a2870078e/akka-actor_2.11-2.3.3.jar
C:/Users/t-danesc/.gradle/caches/modules-2/files-2.1/com.typesafe/config/1.2.1/f771f71fdae3df231bcd54d5ca2d57f0bf93f467/config-1.2.1.jar
C:/Users/t-danesc/.gradle/caches/modules-2/files-2.1/org.scala-lang/scala-actors-migration_2.11/1.1.0/dfa8bc42b181d5b9f1a5d0147f8ae308b893eb6f/scala-actors-migration_2.11-1.1.0.jar

```

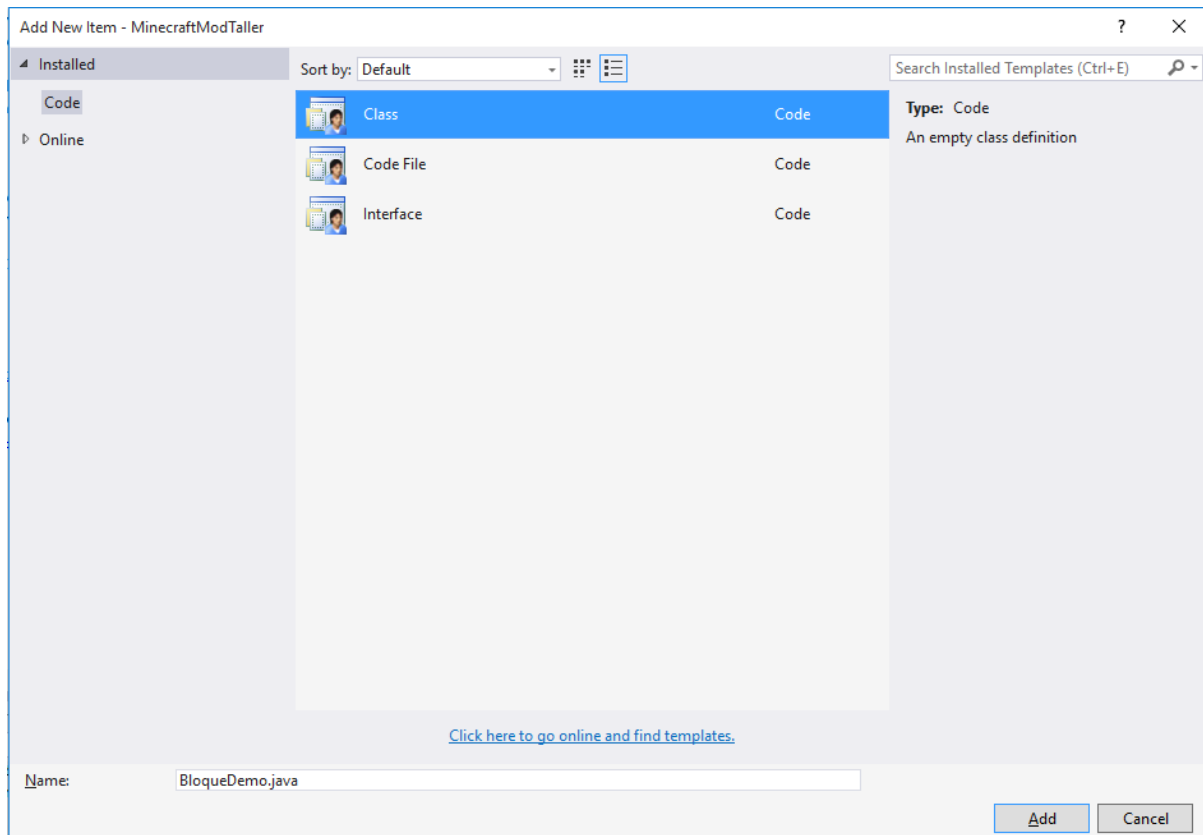
Construcción del bloque personalizado

¡Es hora de construir nuestro bloque!

En primer lugar nos dirigimos a la carpeta *Java* → *com*, y borramos la carpeta *example*.



Crearemos una carpeta denominada *bloque*, y dentro meteremos el código fuente. Crearemos un fichero denominado *BloqueDemo.java*. Para ello hacemos click derecho en la carpeta, y vamos a *Add* → *New Item...* y escogemos *Class*.



Cuando lo tengamos, importaremos los siguientes paquetes:

```
import net.minecraft.block.Block;  
import net.minecraft.block.BlockTNT;  
import net.minecraft.block.material.Material;  
import net.minecraft.creativetab.CreativeTabs;  
import net.minecraftforge.fml.common.registry.GameRegistry;  
import net.minecraft.world.World;  
import net.minecraft.world.Explosion;  
import net.minecraft.util.BlockPos;  
import net.minecraft.block.state.BlockState;  
import net.minecraft.block.state.IBlockState;  
import net.minecraft.entity.player.EntityPlayer;  
import net.minecraft.item.Item;  
import net.minecraft.util.EnumFacing;  
import net.minecraft.init.Items;  
import net.minecraft.client.Minecraft;
```

La clase que creemos deberá heredar de *Block*. Cuando creemos el constructor, deberemos inicializar el de la clase padre con un material que nosotros le demos al bloque. Hay diversos materiales, pero nosotros nos decantaremos por la roca.

Posteriormente le daremos un nombre e indicaremos que aparezca en el apartado de bloques cuando estemos jugando en modo creativo.

La clase quedaría de la siguiente manera:

```
package bloque;

import net.minecraft.block.Block;
import net.minecraft.block.material.Material;
import net.minecraft.creativetab.CreativeTabs;
import net.minecraftforge.fml.common.registry.GameRegistry;
import net.minecraft.world.World;
import net.minecraft.util.BlockPos;
import net.minecraft.block.state.BlockState;
import net.minecraft.block.state.IBlockState;
import net.minecraft.entity.player.EntityPlayer;
import net.minecraft.item.Item;
import net.minecraft.util.EnumFacing;
import net.minecraft.init.Items;
import net.minecraft.client.Minecraft;

public class BloqueDemo extends Block
{
    private final String name = "bloqueDemo";

    public BloqueDemo()
    {
        super(Material.rock);
        setUnlocalizedName(name);
        setCreativeTab(CreativeTabs.tabBlock);
    }

    public String getName()
    {
        return name;
    }
}
```

Los métodos que pueden añadirse a los bloques son:

setHardness

Sirve para configurar la dureza del bloque. Acepta un *float* donde 0.5 sería equivalente a un bloque de arena y 50 sería equivalente a un bloque de obsidiana. A mayor dureza, se necesitan mejores ítems para lograr destruir el bloque.

setStepSound

Se utiliza para asociar un sonido al bloque.

setUnlocalizedName/setBlockName

Se utiliza para dar un nombre interno al bloque. Este nombre se queda en un estado “deslocalizado”. Forge lo localizará automáticamente.

setCreativeTab

Se utiliza para escoger en qué panel aparecerá el bloque dentro del modo creativo.

Ahora mismo tenemos nuestro bloque definido, pero nos quedan varias cosas. En primer lugar, tenemos que registrar el bloque en el mod, para que Forge lo reconozca.

En la raíz de la carpeta *java*, creamos una nueva clase con el nombre de *TallerMod.java*. Será nuestra clase base del mod, y en la cual se registrarán el resto de clases que formen parte de éste.

```
import net.minecraft.init.Blocks;
import net.minecraftforge.fml.common.Mod;
import net.minecraftforge.fml.common.Mod.EventHandler;
import net.minecraftforge.fml.common.event.FMLInitializationEvent;
import net.minecraft.block.Block;
import net.minecraft.block.material.Material;
import net.minecraft.creativetab.CreativeTabs;
import net.minecraftforge.fml.common.registry.GameRegistry;
import net.minecraft.client.renderer.entity.RenderItem;
import net.minecraftforge.fml.relauncher.Side;
import net.minecraft.client.Minecraft;
import net.minecraft.item.Item;
import net.minecraft.client.resources.model.ModelResourceLocation;
import bloque.BloqueDemo;

@Mod(modid=TallerMod.MODID, name=TallerMod.MODNAME, version=TallerMod.MODVER)
public class TallerMod {

    public static final String MODID = "TallerMod";
    public static final String MODNAME = "Taller Mod";
    public static final String MODVER = "1.0";

    public static Block bloqueDemo;

    @EventHandler
    public void init(FMLInitializationEvent event)
    {
        bloqueDemo = new BloqueDemo();
        GameRegistry.registerBlock(bloqueDemo, "bloqueDemo");
    }
}
```

Veamos en qué consiste la clase:

En primer lugar, nos encontramos con la anotación `@Mod`. Existen diferentes anotaciones, que son las siguientes:

`@Mod`

Le dice a Forge que esta es la clase base del mod. Tiene 3 parámetros

- Modid: Nombre identificador del mod. No debería cambiarse nunca tras la primera release del mod.
- Name: Nombre legible del mod.
- Version: La versión del mod.

`@NetworkMod`

Indica a Forge el comportamiento cuando el cliente o el servidor tiene el mod instalado. Tiene 2 parámetros:

- `clientSideRequired`: Pregunta si necesitas tener esto en el cliente para usar el mod. En general es *true*.

@Instance

Las clases base para los mods son Singletons. Esta es la referencia a tu clase que Forge utiliza. Asegúrate de que el argumento es el *modid* que has puesto en @Mod. De lo contrario, tendrá un string vacío por defecto y puede causar conflictos con otros mods que hagan lo mismo.

@SidedProxy

Este proxy es donde Forge decide qué clase cargará dependiendo de si el cliente o el servidor están ejecutándose o no. Se puede usar esta anotación en cualquier campo.

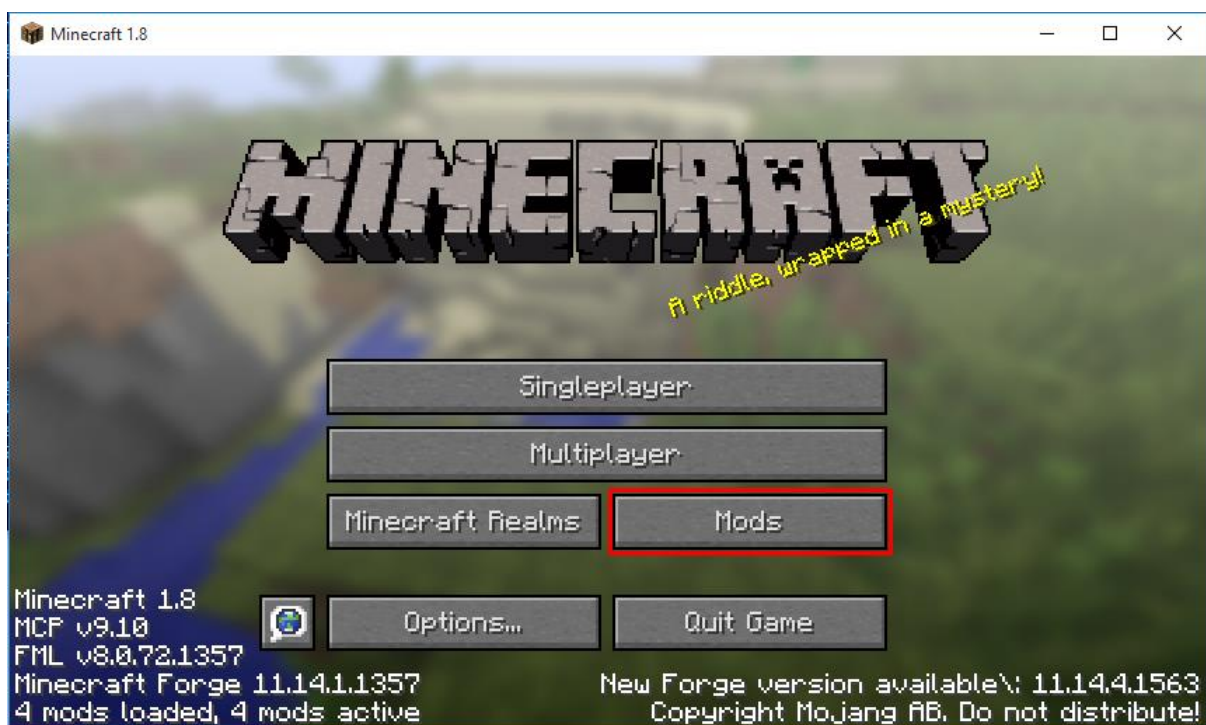
- `clientSide`: String que contiene el nombre complejo de la clase a cargar por el cliente.
- `serverSide`: String que contiene el nombre complejo de la clase a cargar por el servidor.

@EventHandler

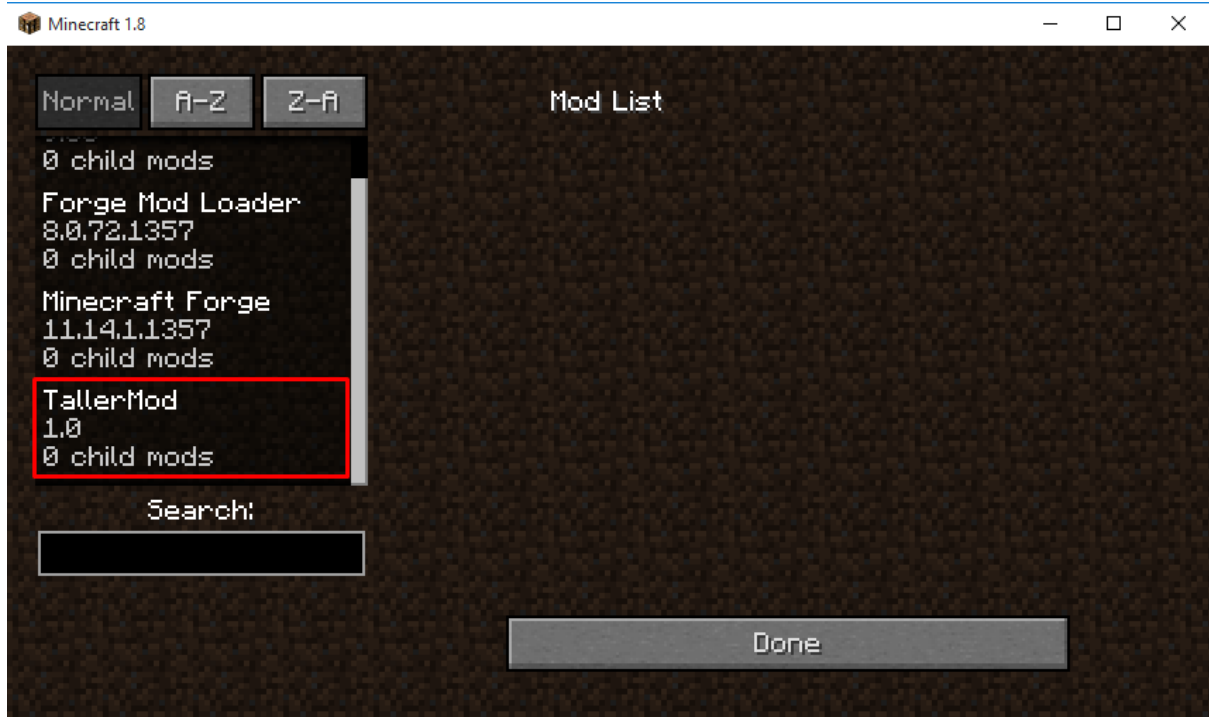
Esta anotación para que Forge sepa cuándo llamar a los métodos.

Cuando Forge cargue la clase base del mod, llamará al método *init* y ejecutará las líneas que se encargan de registrar el bloque que hemos creado.

Si hemos llegado hasta aquí, es hora de ir probando lo realizado. Vamos a ver que al menos Minecraft detecte el mod que estamos realizando. Pulsamos *CTRL + F5*, de manera que ejecutaremos sin depurar (ya que depurando nos va a ir muy lento) y automáticamente iniciará Minecraft tras unos instantes. Cuando cargue, vamos a *Mods*:



Y, si todo ha ido bien, aparecerá nuestro mod en el listado de mods cargados:



Vamos a ver que aparece en el juego si ahora mismo entráramos. Como hemos visto, el mod consiste en un bloque que aparecerá cuando se juegue en modo creativo. Por tanto, vamos a crear un nuevo mundo. Desde el menú principal del juego, vamos a *Singleplayer* y posteriormente a *Create new world*. Tras ello, le damos un nombre al mundo y escogemos modo creativo.



Si ahora nos dirigimos a la pestaña de bloques, y bajamos abajo del todo...

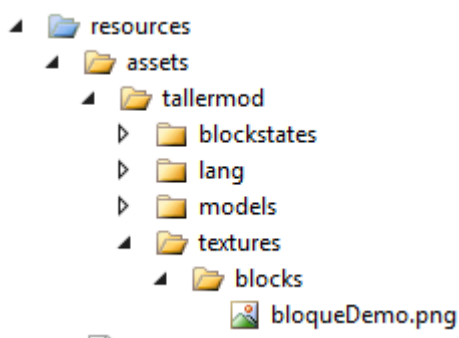


¡Vemos nuestro bloque! Podremos ponerlo en nuestro inventario y empezar a utilizarlo.

Personalización del bloque

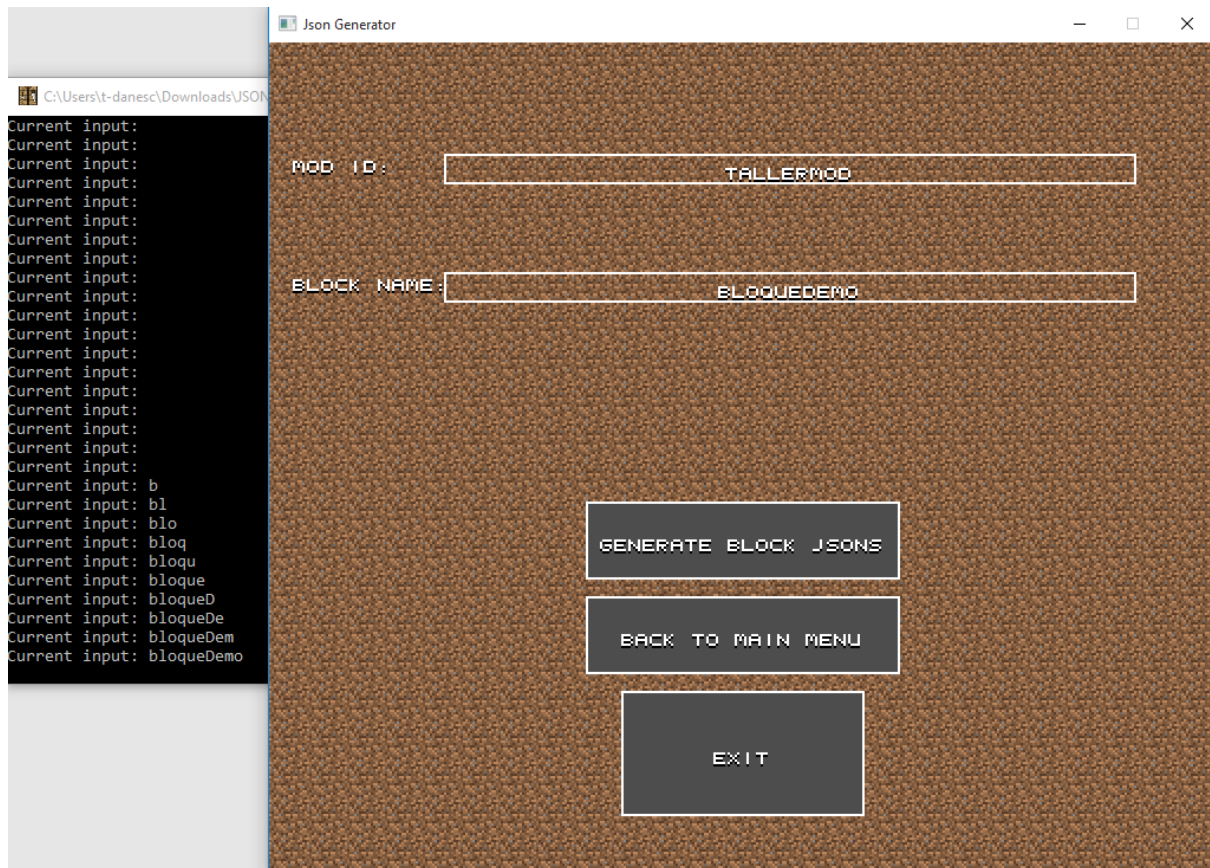
No obstante, el bloque no tiene textura, ni nombre... es un poco feo. Vamos a darle un poco de vida.

Para crear una textura, vamos a utilizar una imagen de 16 x 16 píxeles. En el mismo Paint la podemos crear. Cuando la tengamos, la metemos en la siguiente ruta del proyecto, con el mismo nombre que el nombre que pusimos al bloque (vamos creando las carpetas que no tengamos): *main >> resources >> assets >> [MOD_ID todo minúsculas] >> textures >> blocks*.





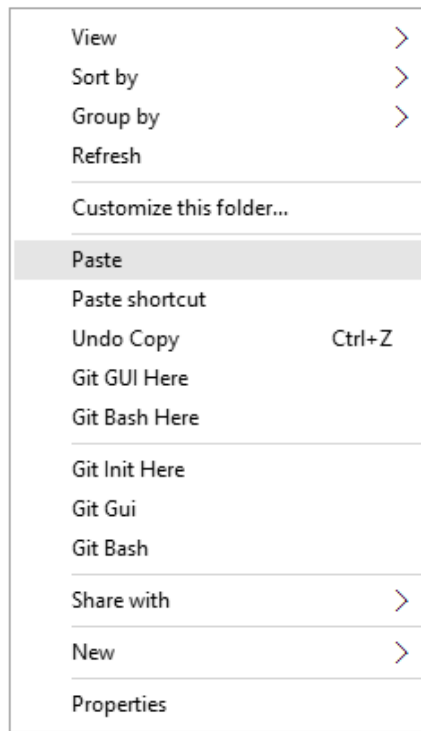
A continuación, tendremos que crear unos ficheros JSON de configuración para nuestro bloque. Para facilitar esta tarea, la comunidad ha creado un programa con el fin de generarlos. Podemos descargarlo [aquí](#).

Lo descomprimos, lo ejecutamos, y le damos a *Block*. Dentro, nos pedirá el ID del Mod y el nombre del bloque, ambos puestos al construir las clases. Aunque no se vean las mayúsculas en el programa, es importante escribirlas, ya que en el JSON si se verán reflejadas.



Ahora nos vamos a la carpeta *output* que habrá generado el programa en el directorio en el que lo hayamos ejecutado, copiamos la carpeta *assets* y la pegamos en el directorio de nuestro proyecto para sobrescribirla.

Documents > Visual Studio 2013 > Projects > MinecraftModTaller > MinecraftModTaller > src > main > resources				
Name	Date modified	Type	Size	
 assets	08/03/2016 15:02	File folder		
 mcmod.info	01/03/2016 11:32	INFO File	1 KB	



Si no vemos reflejados los cambios en Visual Studio, lo reiniciamos y ya aparecerán correctamente.

Para que termine de aparecer correctamente la textura, nos falta escribir unas líneas de código. Dentro de nuestra clase principal del mod, en el mismo método *init*, registramos los renderizados:

```
@EventHandler
public void Init(FMLInitializationEvent event)
{
    bloqueDemo = new BloqueDemo();
    GameRegistry.registerBlock(bloqueDemo, "bloqueDemo");

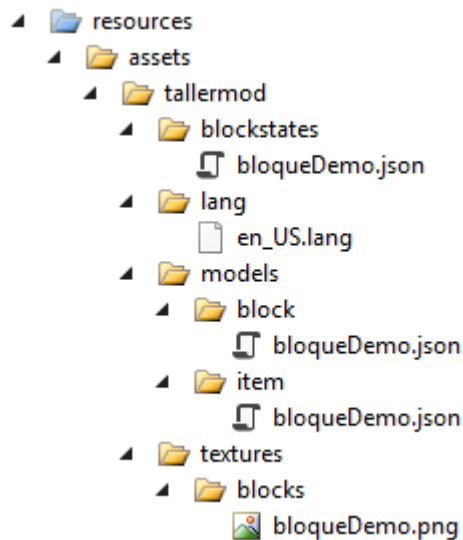
    if(event.getSide() == Side.CLIENT)
    {
        RenderItem renderItem = Minecraft.getMinecraft().getRenderItem();
        renderItem.getItemModelMesher().register(Item.getItemFromBlock(bloqueDemo), 0, new ModelResourceLocation(TallerMod.MODID + ":" + ((BloqueDemo)bloqueDemo).getName(), "inventory"));
    }
}
```

¡Listo! Tenemos nuestra textura para el bloque. Ahora nos falta darle un nombre amigable que aparezca cuando vayamos a seleccionarlo en el listado de bloques.

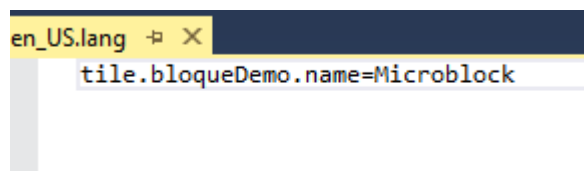
Renombrado del bloque

Para ello, dentro de *assets* >> [*MOD_ID* todo minúsculas], creamos una carpeta llamada *lang* y dentro creamos un fichero *en_US.lang*, ya que en nuestro caso utilizaremos el idioma inglés del juego.

La estructura del proyecto quedará de la siguiente manera:



Dentro del fichero, ponemos lo siguiente:



Y, si ahora volvemos a ejecutar:



¡Nos aparecerá nuestro bloque con la textura correctamente cargada!

Si no nos aparece la textura cargada es porque posiblemente nos haya bailado alguna mayúscula o minúscula de los nombres. Es muy importante que la carpeta con el nombre del ID del mod tenga el nombre con todo minúsculas. Por otro lado, es importante también que el mismo bloque que hayamos dado al bloque quede reflejado en todos los JSON y en la imagen que servirá como textura.

Recetas

¿Qué tal si ahora añadimos una receta? Las recetas indican los bloques que necesitamos para obtener un determinado ítem u otro bloque. Por ejemplo, si queremos obtener un pico de madera, deberemos colocar un determinado número de bloques de madera de una forma determinada. El seguir estas recetas se denomina *crafting*.

Crear recetas es sencillo. Nuestro objetivo va a ser que, colocando 5 bloques de tierra, obtengamos el bloque que acabamos de crear, como se puede observar en la siguiente imagen:



Para ello, de nuevo, en el método *init*, añadiremos el siguiente código:

```
@EventHandler
public void init(FMLInitializationEvent event)
{
    bloqueDemo = new BloqueDemo();
    GameRegistry.registerBlock(bloqueDemo, "bloqueDemo");

    GameRegistry.addRecipe(new ItemStack(bloqueDemo),
        "AAA",
        "A A",
        " ",
        'A', Blocks.dirt);

    if(event.getSide() == Side.CLIENT)
    {
        RenderItem renderItem = Minecraft.getMinecraft().getRenderItem();
        renderItem.getItemModelMesher().register(item.getItemFromBlock(bloqueDemo), 0, new ModelResourceLocation(TallerMod.MODID + ":" + ((BloqueDemo)bloqueDemo).getName(), "inventory"));
    }
}
```

```
GameRegistry.addRecipe(new ItemStack(bloqueDemo),
    "AAA",
    "A A",
    " ",
    'A', Blocks.dirt);
```

Quiere decir lo siguiente:

Con el método *addRecipe* indicamos que queremos añadir una nueva receta. Posteriormente crearemos un *ItemStack*, que tiene como primer parámetro un objeto tipo *Block* o *Item*. Este primer parámetro será el resultado de la receta.

El segundo parámetro es un array. A la hora de *craftear*, podemos hacerlo usando un tablero de 3x3 o de 2x2, en función de si utilizamos una mesa de *crafting* o no. Así pues, los primeros elementos del array serán 3 o 2 strings, con una longitud de 3 o 2.

Para nuestro ejemplo, como se ve en la imagen, necesitaremos colocar 5 bloques de arena con forma de arco para obtener nuestro bloque de Microsoft.

El primer string tendrá 3 letras, podemos usar la que nosotros queramos. En nuestro caso, será la A. Esa letra identificará un tipo de bloque que le especificaremos más adelante al método. Si utilizaríamos otro tipo de bloques para la receta además de arena, utilizaríamos otra letra para identificar al siguiente bloque.

El segundo string tendrá dos letras, solamente. Una en la primera posición y otra en la última. Como en el centro de la segunda fila no queremos ningún bloque, pondremos un espacio.

El tercer string, como no va a representar ningún bloque, tendrá tres espacios.

Para finalizar, indicaremos al método a qué bloque representa cada letra. Para ello, pondremos la letra entre **comillas simples** y seguidamente el tipo de bloque. Si hubiese más bloques, pondríamos [LETRA], [BLOQUE], [LETRA], [BLOQUE], etc.

La próxima vez que inicies el juego, crea un nuevo mundo, esta vez en modo supervivencia. Como hemos hecho una receta de 3x3, necesitarás una mesa de *crafting*. Cuando uses los bloques de arena en la forma indicada, ¡aparecerá el bloque de Microsoft que hemos creado antes!

Aún hay más

Esto es solo el comienzo. Nosotros hemos tocado las bases, pero los mods van mucho más allá. Aquí te dejamos unos recursos para que puedas seguir investigando:

- Tutoriales oficiales de Minecraft Forge: www.minecraftforge.net/wiki/Tutorials
- Foros oficiales de Minecraft – Sección de Mods:
<http://www.minecraftforge.net/forums/mapping-and-modding/minecraft-mods>
- Listado de mods compatibles con Forge: <http://modlist.mcf.li/>